

biblio: consultazione di biblioteche distribuite

Progetto del modulo di laboratorio dei corsi di SO A/B 2011/12

Indice

1	Introduzione	1
1.1	Materiale in linea	2
1.2	Struttura del progetto e tempi di consegna	2
1.3	Valutazione del progetto	2
2	Il progetto: biblio	3
3	Il server	3
4	Il client	5
5	Protocollo di interazione	5
5.1	Formato dei messaggi	5
5.2	Messaggi da Client a Server	6
5.3	Messaggi da Server a Client	6
6	Lo script bibaccess ed il formato del file di log	6
7	Istruzioni	7
7.1	Materiale fornito dai docenti	7
7.2	Cosa devono fare gli studenti	8
8	Parti Opzionali	8
9	Codice e documentazione	8
9.1	Vincoli sul codice	8
9.2	Formato del codice	9
9.3	Relazione	9

1 Introduzione

Il modulo di Laboratorio di Programmazione di Sistema del corso di Sistemi Operativi e Laboratorio (277AA) prevede lo svolgimento di un progetto individuale suddiviso in tre frammenti. Questo documento descrive la struttura complessiva del progetto e dei vari frammenti che lo compongono.

Il progetto consiste nello sviluppo del software relativo a **biblio**: un sistema che permette l'interrogazione di un insieme di biblioteche distribuite. Il software viene sviluppato e documentato utilizzando gli strumenti, le tecniche e le convenzioni presentati durante il corso.

1.1 Materiale in linea

Tutto il materiale relativo al corso può essere reperito sul sito Web:

<http://www.cli.di.unipi.it/doku/doku.php/informatica/sol/laboratorio12/>

Il sito verrà progressivamente aggiornato con le informazioni riguardanti il progetto (es. FAQ, suggerimenti, avvisi), il ricevimento e simili. Il sito è un Wiki e gli studenti possono registrarsi per ricevere automaticamente gli aggiornamenti delle pagine che interessano maggiormente. In particolare consigliamo a tutti di registrarsi alla pagina delle 'FAQ' e degli 'Avvisi Urgenti'.

Eventuali chiarimenti possono essere richiesti consultando i docenti di del corso durante l'orario di ricevimento, le ore in laboratorio e/o per posta elettronica.

1.2 Struttura del progetto e tempi di consegna

Il progetto deve essere sviluppato dallo studente individualmente e può essere consegnato entro il 31 Gennaio 2013.

La consegna del progetto avviene *esclusivamente* inviando per posta elettronica il tar creato dal target **consegna3** del **Makefile** contenuto nel kit di sviluppo del progetto. Il tar deve essere allegato ad un messaggio con soggetto

lso12: Consegna Terzo Frammento

ed inviato a susanna@di.unipi.it. Le consegne sono seguite da un messaggio di conferma da parte del docente all'indirizzo di mail da cui la consegna è stata effettuata. Se la ricezione non viene confermata entro 3/4 giorni lavorativi, contattare il docente per e-mail.

*I progetti che non rispettano il formato o non generati con il target **consegna3** non verranno accettati. Si prega di controllare che tutti i file necessari alla corretta compilazione ed esecuzione del progetto siano presenti nel tar prima di inviarlo.*

La data ultima di consegna è il 31/01/2013. Dopo questa data gli studenti dovranno svolgere il nuovo progetto previsto per il corso 2012/13.

Inoltre, il progetto è suddiviso in tre frammenti. Gli studenti che consegnano una versione sufficiente di ogni frammento entro la data di scadenza specificata sul WEB accumulano dei bonus di 2 punti che contribuiscono al voto finale con le modalità illustrate nelle slide introduttive del corso (vedi sito).

1.3 Valutazione del progetto

Al progetto viene assegnata una valutazione da 0 a 30 di cui 6 punti esclusivamente determinati dalla *qualità della documentazione allegata*. La valutazione del progetto è effettuata in base ai seguenti criteri:

- motivazioni, originalità ed economicità delle scelte progettuali
- strutturazione del codice (suddivisione in moduli, uso di makefile e librerie etc.)
- efficienza e robustezza del software
- modalità di testing
- aderenza alle specifiche

- qualità del codice C e dei commenti
- chiarezza ed adeguatezza della relazione (vedi Sez. 9.3)

La prova orale tenderà a stabilire se lo studente è realmente l'autore del progetto consegnato e verterà su tutto il programma del corso e su tutto quanto usato nel progetto anche se non fa parte del programma del corso. Il voto dell'orale (da 0 a 30L) fa media con la valutazione del progetto per delineare il voto finale. In particolare, l'orale comprenderà

- una discussione delle scelte implementative del progetto e dei frammenti
- l'impostazione e la scrittura di script bash e makefile
- l'impostazione e la scrittura di programmi C + PosiX non banali (sia sequenziali che concorrenti)
- domande su tutto il programma presentato durante il corso.

Casi particolari Gli studenti lavoratori iscritti alla laurea triennale in Informatica possono consegnare i due frammenti e il progetto finale in un'unica soluzione in qualsiasi momento dell'anno ed essere valutati con votazione da 0 a 30. In questo caso è necessaria la certificazione da consegnare al docente come da regolamento di ateneo.

Gli studenti che svolgono il progetto per le lauree di secondo livello sono invitati a contattare il docente.

Gli studenti iscritti ai vecchi ordinamenti (nei quali il voto di Lab. 4 contribuiva al voto di SO) avranno assegnato un voto in trentesimi che verrà combinato con il voto del corso di SO corrispondente secondo modalità da richiedere ai due docenti coinvolti.

2 Il progetto: biblio

Lo scopo del progetto è lo sviluppo di un sistema di interrogazione per la ricerca di testi in un insieme di biblioteche distribuite. Il catalogo di ogni biblioteca è gestito da un server, interrogabile su una socket di nome noto.

Il sistema è costituito da più istanze dei seguenti processi:

- **bibclient**: il processo client (uno per ogni interrogazione) che si occupa di interagire con i server per ottenere informazioni sui volumi e richiederne il prestito;
- **bibserver**: il processo server che gestisce il catalogo di una sigola biblioteca, ovvero: verifica se un volume è presente nel catalogo, fornisce le informazioni relative al volume e (se possibile) ne registra il prestito.

bibclient e **bibserver** sono i due processi che devono essere realizzati nel progetto didattico. I processi comunicano via socket AF_UNIX. Per rendere più semplice lo sviluppo ed il testing dell'applicazione sulle macchine del centro di calcolo per la didattica tutte le socket vengono create nella directory locale `./tmp` invece che in `/tmp` come sarebbe logico aspettarsi. Quest'ultima soluzione renderebbe infatti possibili interazioni indesiderate e fastidiose fra progetti sviluppati da utenti diversi sulla stessa macchina.

Inoltre, deve essere realizzato uno script bash (**bibaccess**) che analizza il file di log in cui ogni server registra le richieste processate come specificato in Sez. 6.

3 Il server

bibserver viene attivato da shell con il comando

```
$ bibserver name_bib file_record
```

dove **file_record** è il file di testo che contiene la registrazione delle schede relative a tutti i volumi della biblioteca **name_bib**. Un esempio di formato di tali record è il seguente

```
autore: Kernighan, Brian W.; autore: Ritchie, Dennis M.; titolo: Il linguaggio  
C (seconda Edizione); editore: Jackson Libri; anno: 1989; nota:  
Edizione italiana; collocazione: Z.22.56; luogo_pubblicazione: Milano;  
descrizione_fisica: 359 p., softcov, 13 cm;
```

i record sono separati da un `\n` (ritorno carrello) ogni record è costituito da un numero variabile di campi che descrivono autore/i, titolo, editore, anno, etc. Ogni campo ha il formato

```
nomecampo: valore1 , .... , valoreN ;
```

in cui **nomecampo** è l'etichetta del campo (deve appartenere ad un insieme fissato) e **valore1..valoreN** sono uno o più valori associati a quell'etichetta. Se i valori sono più di uno sono separati da virgola (,). Le informazioni di un campo sono terminate da punto e virgola (;). Il formato dei valori per ogni etichetta è specificato nel file **bib.h**, ad esempio è possibile avere più campi autori e per ogni campo autore ci sono due valori. Il primo che descrive il cognome dell'autore ed il secondo che descrive il nome. I valori sono sempre stringhe di lunghezza limitata. I limiti per ciascuno campo sono specificati in **bib.h**.

Il server registra anche un file di log (`./name_bib.log`) in cui per ogni richiesta effettuata da un client si registrano il numero dei record inviati, dei prestiti effettuati e le informazioni relative a ciascun record secondo il formato in Sez. 6.

Il funzionamento del server è il seguente. Se il file passato come argomento esiste viene aperto in lettura e ne viene verificato il formato. Se il formato è corretto viene creato il file di log (se il file già esiste viene troncato) e viene creata una socket **AF_UNIX** di indirizzo

```
./tmp/name_bib.sck
```

se la socket esiste già l'avvio, il server fallisce. Su questa socket i client apriranno le connessioni con i server delle varie biblioteche per inviare le richieste di informazioni/prestito.

Il server è multithreaded in quanto ogni richiesta viene servita da un thread **worker** in parallelo con l'accettazione di connessioni da parte di altri client. I worker lavorano su una struttura condivisa che contiene le informazioni su tutte le schede contenute nel **file_record**. Se il client richiede un prestito, questo viene concesso soltanto se il volume non è già stato prestato. Ogni prestito dura due mesi. Nel calcolo della data di scadenza, se il giorno del rientro viene calcolato avanzando di 2 il mese a partire dalla data corrente. Se la data ottenuta non esiste (es 30-02-12) si passa al primo giorno del mese successivo. I prestiti vengono registrati in RAM nella struttura condivisa fra i thread. Alla chiusura, il server deve aggiornare **file_record** sovrascrivendolo con i record aggiornati della struttura condivisa.

Ogni client invia una sola richiesta. Il server, per ogni connessione, attiva un thread worker che si occupa leggere la richiesta del worker scandire la struttura condivisa alla ricerca di uno o più record che verificano la richiesta effettuata ed eventualmente registrare il prestito. Le risposte alle richieste fatte verranno inviate al client sulla stessa socket di connessione secondo il formato specificato in Sez. 5. Le richieste del client possono essere di due soli tipi. Una *query*, in cui si chiedono i record che contengono una specifica stringa in uno o più campi e un *loan* (prestito) in cui si richiede anche il prestito di tutti i volumi relativi a record che verificano la query.

Il server termina quando riceve un segnale di **SIGINT** o **SIGTERM**, in questo caso si attende la terminazione dei thread worker, in modo che vengano elaborate le richieste

pendenti, si termina la scrittura del file di log, si registra il nuovo `file_record` e si elimina la socket del server. Infine il server viene terminato.

Il protocollo di interazione fra client e server è descritto nella Sezione 5.

4 Il client

Il client `bibclient` è un comando Unix che viene attivato da linea di comando per attivare la ricerca delle informazioni relative ad un record che contiene particolari stringhe. Il formato di una invocazione è

```
$ ./bibclient --author="ciccio" -p
$ ./bibclient --author="ciccio" --title="pippo"
```

dove ogni opzione con doppio meno corrisponde al nome di un possibile campo del record, mentre l'opzione `-p` serve a richiedere il prestito. Per ogni invocazione è possibile specificare solo una opzione lunga (con doppio meno) per ogni campo. Il valore specificato dopo il simbolo `=` è una stringa che deve essere **contenuta** nel campo specificato. La risposta è negativa, se non esiste alcun record che contiene le stringhe specificate nei corrispondenti campi e positiva altrimenti. In caso di risposta positiva il client riceve dal server tutti i record che verificano le condizioni secondo il formato della registrazione su file nel primo frammento di progetto. Ogni record ricevuto è stampato su stdout. Se l'opzione `-p` è specificata viene comunque richiesto il prestito di tutti i volumi che verificano le condizioni, a patto che non siano già in prestito presso altri utenti della biblioteca. Il prestito viene registrato con durata pari a due mesi a partire dall'istante della interrogazione con le modalità di aggiornamento della data specificate per il server.

All'avvio, `bibclient` effettua il parsing delle opzioni e, se l'invocazione è corretta, legge il file `bib.conf` che contiene tutti i nomi (`bib_name`) delle biblioteche disponibili per l'interrogazione. Poi ciclicamente interroga tutte le biblioteche connettendosi sulla socket `./tmp/bib_name.sck` secondo il protocollo specificato nella Sez. 5. Ogni biblioteca viene interrogata per `NTRIAL` tentativi a distanza di `NSEC` secondi. entrambe queste costanti sono specificate nel file `comsock.h`.

5 Protocollo di interazione

I server ed i client interagiscono con un socket `AF_UNIX`.

I client si connettono al server della biblioteca `bib_name` attraverso la socket pubblica `./tmp/bib_name.sck`, creata all'avvio del server. I tipi descritti in questa sezione e i prototipi delle funzioni di comunicazione da realizzare per il terzo frammento di progetto sono contenuti nel file `comsock.h`.

5.1 Formato dei messaggi

I messaggi scambiati fra server e client hanno la seguente struttura:

```
typedef struct {
    char type;
    unsigned int length;
    char buffer[MAXBUF];
} message_t;
```

dove `MAXBUF` è definita in `comsock.h`. Nella realizzazione delle funzioni di comunicazione deve essere gestito correttamente l'invio di messaggi eventualmente più lunghi di `MAXBUF`.

Il campo **type** è un **char** (8 bit) che contiene il tipo del messaggio spedito. **type** può assumere i seguenti valori:

```
#define MSG_QUERY      'Q'
#define MSG_LOAN       'L'
#define MSG_RECORD     'R'
#define MSG_NO         'N'
#define MSG_ERROR      'E'
```

Il campo **length** è un **unsigned int** che indica il numero dei dati significativi all'interno del campo **buffer**. Il campo **length** vale 0 nel caso in cui il campo **buffer** non sia significativo. Per ogni **send**/**receive** si richiede di inviare solo i byte significativi del messaggio e non tutto il campo **buffer**.

5.2 Messaggi da Client a Server

Nei messaggi spediti dal client al Server, il campo **type** può assumere i seguenti valori:

MSG_QUERY Messaggio per richiedere i record che contengono alcune parole specifiche in alcuni campi. Il campo **buffer** contiene le condizioni da verificare secondo il formato

```
campo1: valore1;... campoN: valoreN;
```

Il server risponderà con un **MSG_RECORD** se esiste almeno un record che soddisfa la richiesta e invierà i record completi in una serie successiva di messaggi di tipo **MSG_RECORD**. Se nessun record soddisfa la query risponderà con **MSG_NO**, mentre se si è verificato un errore risponderà con **MSG_ERROR** ed una eventuale stringa che descrive l'errore.

MSG_LOAN Messaggio per richiedere il prestito di tutti i record che contengono alcune parole specifiche in alcuni campi. Funziona esattamente come **MSG_QUERY** ma seleziona il prestito dei volumi disponibili che verificano la query. In questo caso vengono inviati al client solamente i record per cui è stato accordato il prestito.

5.3 Messaggi da Server a Client

Nei messaggi spediti dal Server a Client, il campo **type** può assumere i seguenti valori:

MSG_ERROR Messaggio di errore. Questo tipo di messaggio viene spedito quando si è verificato un errore nel processare la richiesta del client. Il campo **buffer** può contenere una stringa che spiega l'errore verificatosi.

MSG_NO Messaggio di risposta negativa. Con questo messaggio il server segnala che non ci sono record che verificano la query.

MSG_RECORD Messaggio di invio record. Il campo **buffer** contiene il record completo come stringa correttamente terminata da **\0**.

6 Lo script bibaccess ed il formato del file di log

Ogni richiesta processata dal server viene registrata all'interno di un file di log. All'inizio dell'alaborazione il server tronca il file (ne elimina eventuali contenuti precedenti) poi per ogni richiesta di tipo **MSG_QUERY** registra delle linee del tipo

```
QUERY 0
```

se nessun record verifica la query oppure

```
autore: Luccio, Fabrizio; autore: Pagli, Linda; autore: Steel, Graham; titolo: M
athematical and Algorithmic Foundations of the Internet; editore: CRC Press, Tay
lor and Francis Group; luogo_pubblicazione: New York; anno: 2011; collocazione:
Z.DDf.56;descrizione_fisica: 434 p., softcov, 22 cm;nota: Chapman & Hall/CRC Ap
plied Algorithms and Data Structures series;
QUERY 1
```

ovvero tutti i record inviati, nel formato specificato in `bib.h`, seguiti da una linea che contiene solo `QUERY` ed il numero totale di record inviati. Non è richiesto che tutte le linee relative ai record che corrispondono ad una singola query siano contigue nel file. Il formato per il prestito è analogo

```
autore: Luccio, Fabrizio; autore: Pagli, Linda; autore: Steel, Graham; titolo: M
athematical and Algorithmic Foundations of the Internet; editore: CRC Press, Tay
lor and Francis Group; luogo_pubblicazione: New York; anno: 2011; prestito: 20-0
3-2012; collocazione: Z.DDf.56;descrizione_fisica: 434 p., softcov, 22 cm;nota:
Chapman & Hall/CRC Applied Algorithms and Data Structures series;
LOAN 1
```

se è stato possibile accordare il prestito (notare la presenza della data di scadenza), oppure

```
LOAN 0
```

se non c'è stato prestito.

`bibaccess` è uno script bash che elabora off-line i file di log generati dai `bibserver`. Lo script può essere invocato con due opzioni:

```
$ ./bibaccess --query log1 ... logN
$ ./bibaccess --loan log1 ... logN
```

dove `log1...logN` sono file di testo (ASCII) che contengono i log dei record prestati dai vari server, mentre con le opzioni precedute da doppio trattino si `--query` e `--loan` si può richiedere il numero complessivo di richieste e di prestiti presenti nei file specificati. L'ordine di opzioni e parametri può essere qualsiasi.

Lo script deve controllare la validità dei suoi argomenti, scorrere i file di log ed individuare le informazioni richieste. All'fine dell'elaborazione stampa le parole `QUERY` o `LOAN` e il loro valore complessivo seguito da `Bye`. Si rimanda al README-2 ed ai test del secondo frammento per una descrizione più accurata degli output attesi. È fatto esplicito divieto di usare `sed` o `awk` nella realizzazione dello script.

7 Istruzioni

7.1 Materiale fornito dai docenti

Nei kit del progetto vengono forniti

- funzioni di test e verifica
- `Makefile` per test e consegna
- file di intestazione (`.h`) con definizione dei prototipi e delle strutture dati
- vari README di istruzioni

7.2 Cosa devono fare gli studenti

Gli studenti devono:

- leggere *attentamente* i README dei vari frammenti e capire il funzionamento del codice fornito dai docenti
- implementare le funzioni richieste e testarle
- verificare le funzioni con i test forniti dai docenti (attenzione: questi test vanno eseguiti su codice già corretto e funzionante altrimenti possono dare risultati fourvianti o di difficile interpretazione)
- preparare la *documentazione*: vedi la Sez. 9.
- sottomettere i tre frammenti del progetto esclusivamente utilizzando il makefile fornito e seguendo le istruzioni nel README.

8 Parti Opzionali

Possono essere realizzate funzionalità ed opzioni in più rispetto a quelle richieste (ad esempio altri comandi del client o altre statistiche nello script).

Le parti opzionali devono essere spiegate nella relazione e corredate di test appropriati.

9 Codice e documentazione

In questa sezione, vengono riportati alcuni requisiti del codice sviluppato e della relativa documentazione.

9.1 Vincoli sul codice

Makefile e codice devono compilare ed eseguire CORRETTAMENTE su (un sottinsieme non vuoto del) le macchine del CLI. Il README (o la relazione) deve specificare su quali macchine è possibile far girare correttamente il codice. Inoltre, se si usano software e librerie non presenti al CLI: (1) devono essere presenti nel tar TUTTI i file necessari per l'installazione in locale del/i tool e (2) devono essere presenti nel makefile degli opportuni target per effettuare automaticamente l'installazione in locale. Se questa condizione non è verificata il progetto non viene accettato per la correzione.

La stesura del codice deve osservare i seguenti vincoli:

- la compilazione del codice deve avvenire definendo delle regole appropriate nella parte iniziale del makefile contenuto nel kit;
- il codice deve compilare senza errori o warning utilizzando le opzioni `-Wall -pedantic`
- NON devono essere utilizzate funzioni di temporizzazioni quali le `sleep()` o le `alarm()` per risolvere problemi di race condition o deadlock fra i processi/thread. Le soluzioni implementate devono necessariamente funzionare qualsiasi sia lo scheduling dei processi/thread coinvolti
- DEVONO essere usati dei nomi significativi per le costanti nel codice (con opportune `#define` o `enum`)

9.2 Formato del codice

Il codice sorgente deve adottare una convenzione di indentazione e commenti chiara e coerente. In particolare deve contenere

- una intestazione per ogni file che contiene: il nome ed il cognome dell'autore, la matricola, il nome del programma; dichiarazione che il programma è, in ogni sua parte, opera originale dell'autore; firma dell'autore.
- un commento all'inizio di ogni funzione che specifichi l'uso della funzione (in modo sintetico), l'algoritmo utilizzato (se significativo), il significato delle variabili passate come parametri, eventuali variabili globali utilizzate, effetti collaterali sui parametri passati per puntatore etc.
- un breve commento che spieghi il significato delle strutture dati e delle variabili globali (se esistono);
- un breve commento per i punti critici o che potrebbero risultare poco chiari alla lettura
- un breve commento all'atto della dichiarazione delle variabili locali, che spieghi l'uso che si intende farne

9.3 Relazione

La documentazione del progetto consiste nei commenti al codice e in una breve relazione (massimo 10 pagine) il cui scopo è quello di descrivere la struttura complessiva del lavoro svolto. La relazione *deve rendere comprensibile il lavoro svolto ad un estraneo, senza bisogno di leggere il codice se non per chiarire dettagli implementativi. In particolare NON devono essere ripetute le specifiche contenute in questo documento.* In pratica la relazione deve contenere:

- le principali scelte di progetto (strutture dati principali, algoritmi fondamentali e loro motivazioni)
- la strutturazione del codice (logica della divisione su più file, librerie etc.)
- la struttura dei programmi sviluppati
- la struttura dei programmi di test (se ce ne sono)
- le difficoltà incontrate e le soluzioni adottate
- quanto altro si ritiene essenziale alla comprensione del lavoro svolto
- README di istruzioni su come compilare/eseguire ed utilizzare il codice

La relazione deve essere in formato PDF.